OULU UNIVERSITY OF
APPLIED SCIENCES

Timo Leskelä

DEVELOPING A CUSTOMER ACCOUNT CONTROL SYSTEM FOR A SMALL BUSINESS

DEVELOPING A CUSTOMER ACCOUNT CONTROL SYSTEM FOR A SMALL BUSINESS

Timo Leskelä
Bachelor's Thesis
Spring 2018
Degree Programme in
Business Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Business Information Technology

Author: Timo Antero Leskelä
Title of Bachelor's thesis: Developing a Customer Account Control System for a Small Business
Supervisor: Anu Niva
Term and year of completion: Spring 2018 Number of pages: 41

The thesis was commissioned by the student café at the School of Business and Information Management of the Oulu University of Applied Sciences (OUAS). The goal of the thesis was to develop an information system for a small business which manages customer accounts and transactions. The final product consists of the theoretical portion along with a proof-of-concept system based on the café's needs and requirements. The system was developed using HTML, CSS, PHP, SQL and JavaScript.

The thesis topic came from the cafe wishing to replace their system of pre-paid coffee cards with a computerised system that would link customer accounts to student cards distributed by the café. Co-operation with the café ended during the thesis due delays arising from personal reasons and the change of staff, leadership, and focus that occurred within the organisation. Despite this, it was felt that the thesis had enough merit to continue using the café as an example rather than an actual client. Losing the client meant that restrictions would apply to the availability of hardware necessary for full system implementation. This has been reflected in the thesis by altering the system design accordingly.

The thesis examines the challenges present in software development for small companies. A number of software development life cycle models were evaluated, leading to the Waterfall model being selected. The development of the system was carried out according to the method suggested by the Waterfall model. A proof-of-concept system was created in order to prove the feasibility of the idea.

The system created matches the requirements outlined at the start of development. The original requirements of the café were adjusted after the loss of the client. The thesis gave the opportunity to improve as a developer and gain experience with tools both familiar and new. For future considerations it would be interesting to run a similar project with more client involvement and a different development model.

Keywords: HTML, CSS, PHP, SQL, SME, SDLC, Information System

CONTENTS

1 INTRODUCTION

The thesis was originally commissioned by the student café, Café Carma, of the School of Business and Information Management of the Oulu University of Applied Sciences (OUAS). The café is managed by Oulun Tradenomiopiskelijat Ry (OTRO), an independent student organisation, which works in co-operation with the Student Union of the OUAS (OSAKO). OSAKO is a politically independent service and support organisation for the students of OUAS whose main task is to represent student interests inside the OUAS and in the Oulu region (OSAKO 2018, cited 25.2.2018). OSAKO is a member of University of Applied Sciences Students in Finland (SAMOK) which promotes student interests at a national level. Becoming a member of OSAKO entitles students to either a SAMOK or Frank student card along with the associated benefits. After paying the annual membership fee students receive a sticker which indicates that the card is valid for that academic year. (OSAKO 2017a, cited 25.2.2018)

There are several smaller student associations within OUAS which represent students of a specific field of study. OTRO represents students of business and information management. Along with administering student cards, OTRO also arranges student activities and operates Café Carma. They also act as one of OSAKO's service points together with the other student associations. The Board of OTRO consists of 10-15 members, who are active throughout the year and organise varied activities regularly. (OSAKO 2017b, cited 25.2.2018)

This thesis topic came from Café Carma wishing to replace their pre-paid coffee cards. At that time, the café sold cardboard cards which entitled the bearer to a set amount of coffees. However, the café management felt that the cardboard cards should be phased out. Instead they wanted a computerised system that would link a customer account to the student card provided by OSAKO. The student card would act as an identifier for each customer's account by using a card reader or scanner and the system to manage the account information. The system would store data about customer accounts and remaining coffee purchases in a database.

As such, the goal of this thesis is to develop an information system for a small business which manages customer accounts and transactions. The final product consists of the theoretical portion along with a proof-of-concept system based on the business' needs and requirements. In particular,

this thesis will consider challenges of software development for small businesses, especially restrictions caused by the size and resources of the company.

Unfortunately, co-operation with the café ended during the thesis. This was due to delays arising from personal reasons and the change of staff, leadership, and focus that occurred within OTRO's administration. Despite this, it was felt that the thesis had enough merit to continue using Café Carma as an example rather than an actual client. Losing the client meant that restrictions would apply to the availability of hardware necessary for full system implementation. This has been reflected in the thesis by altering the system design accordingly.

The thesis is split into sections covering the knowledge base and development process. The Small Businesses and Software Development section examines what are small businesses and software development and what are the challenges faced by both. Software Development Life Cycle aims to define the concept and examine several models and their advantages and disadvantages. One model that is found suitable will be picked for use in the thesis. Development Process will describe the development work done for the thesis. It will go through the process according to the chosen software development life cycle model. Discussion and Conclusions will contain an evaluation of the thesis work and discuss the challenges, successes, failures, and lessons learned. The thesis will also consider how the work could be improved or expanded upon.

2 SMALL BUSINESSES AND SOFTWARE DEVELOPMENT

This chapter examines the definition and impact of small businesses and software development and the challenges they face. There may be unique challenges to consider when developing software for small businesses. The aim of the chapter is to introduce the reader to the concepts involved with the thesis and demonstrate why the topic is worth exploring.

2.1 Small Businesses

A business, or enterprise, can be defined as an organisation which produces and sells goods or provides a service (Collins English Dictionary 2018, cited 8.3.2018). An enterprise is 'any entity engaged in an economic activity, irrespective of its legal form' (European Commission 2016, 9). This means if an entity is an enterprise or not is determined by its activity, not whether it is a partnership or sole proprietor. Economic activity is usually seen as the sale of products or services at a given price, on a given/direct market (European Commission 2016, 9). Businesses can be further defined by factors such as their product line, country of operations, and business sector they operate in (Aaker & David 2010, 3). They can be privately owned, non-profit or state-owned (BusinessDictionary.com 2018, cited 16.3.2018).

Businesses can also be categorised according to their size. They can be divided into large, medium and small enterprises. The criteria that determines which group a business belongs to can vary depending on international, local and personal standards. The updated definition of micro, small and medium enterprises, MSMEs, by the European Commission narrows it down to size and resources as the main criteria (European Commission 2016, 4). Size refers to employees, turnover and balance sheet, resources to ownership, partnerships and linkages between companies. Though meeting the employee amount criteria is always necessary to get the MSME definition, of the balance sheet and annual turnover criteria only one needs to be met.

In Finland MSMEs are defined as having fewer than 250 paid employees and whose annual turnover is either not more than 50 million euros, or balance sheet total is not more than 43 million euros. A small enterprise is differentiated from medium enterprises by having fewer than 50 paid employees, an annual turnover either not more than 10 million euros or balance sheet total not

more than 10 million euros. The MSME classification also requires that the enterprise meet the criterion of independence. The criterion states that more than 25 percent of the enterprise equity should not be held by one enterprise or jointly held by such enterprises to which the definition of an MSME or of a small enterprise cannot be applied depending on the circumstances. (Statistics Finland 2018, cited 8.3.2018) The European Commission further defines a micro enterprise when the paid employees are less than 10 and turnover not more than 2 million euros or balance sheet not more than 2 million euros (European Commission 2003, cited 16.3.2018). The criteria are compiled into Table 1.

TABLE 1. SME criteria (European Commission 2016, 11)

Enterprise category	Headcount: Annual Work Unit (AWU)	Annual turnover	Annual balance sheet total
Medium-sized	< 250	≦€50 million	≦ €43 million
Small	< 50	≦€10 million	≦€10 million
Micro	< 10	≦€2 million	≦€2 million

MSMEs can be considered the backbone of an economy. The main argument supporting this statement is the MSMEs contribution as job creators. (Robu 2013, 86) Among the 35-member countries of the Organisation for Economic Co-operation and Development MSMEs account for 60 percent of total employment (OECD 2017, 3). Though direct causality link between MSMEs and economic growth is hard to prove, a thriving MSME sector is common in strong economies (Beck et al. 2005, 4). The importance of MSMEs to an economy seems to be high in economies with high-income and GDP per capita, but the opposite is true in case of low-income economies, when the importance of the informal business sector is high (Ayyagari et al. 2007, 416-417). The informal sector refers to jobs that are not considered normal sources of income and no taxes are paid (BusinessDictionary.com 2018, cited 17.3.2018). In Finland MSMEs form 98,8 percent of all Finnish enterprises and employ 65 percent of all private-sector employees (Yrittäjät.fi 2018, cited 17.3.2018).

MSMEs regularly face challenges to their survival and growth in terms of cost, quality, delivery, flexibility and human resource development. Competitive markets and uncertain survival drive companies to seek a competitive advantage. (Singh et al. 2008, 181) Information technology can

help companies improve their business processes, communication and support their business activities (Barba-Sánchez et al. 2007, 103).

2.2 Software Development

Software is a generic term used to describe computer programs (Techopedia 2018, cited 26.3.2018). It contains computer programs, data structures that enable the programs to manipulate information and descriptive information that describes the operation and use of the programs (Pressman & Maxim 2015, 4). Software can be divided into seven categories: system software, application software, engineering software, embedded software, product-line software, web/mobile applications, and artificial intelligence software (Pressman & Maxim 2015, 7). Software is built by software engineers and is used by nearly everyone in the industrialised world (Pressman & Maxim 2015, 1). A rather unique quality of a software product is that it is not considered finished after the product has been shipped but continues to receive updates and features across its lifespan (Siroky 2016, cited 26.3.2018).

Software development is the activity of creating computer programs (Cambridge English Dictionary 2018, cited 27.3.2018). It encompasses all steps in the products lifecycle from design and documentation to implementation and post-launch support (Rouse 2016 cited 1.5.2018). Software engineering is applying engineering principles to the software development process (Techopedia.com 2018, cited 26.3.2018). The purpose of software engineering is to guarantee that the software being developed is of quality and built according to schedule and includes process, methods and tools to help achieve this (Pressman & Maxim 2015, 15-16, 27).

Software development projects face many challenges. Industry surveys suggest that only 25 percent of software projects are completed as scheduled, budgeted and specified (Brookfield et al., 2014, 197). Up to 80 percent of all software projects go over budget, the average project exceeding the budget by 50 percent (Schnidt, et al. 2001, 6). One study found the lack of commitment from top level management to be considered the most important risk factor by project managers (Schnidt, et al. 2001, 20). This was compounded by another study, with additional factors of unclear budgeting, lack of staff and staff skills, poor management of project changes, failure to satisfy end-user expectations and ignored requirements for the sake of technology being the highest rated risks (Tesch, et al. 2007, 64).

Another study isolated nine risk components consisting of 45 risk factors that are most likely to influence project failure: project user engagement, technology failure, project personnel, technology and system requirements, project implementation, project planning, feasibility study, project process, and feasibility study decision. Of these, risk factors related to project user engagement, technology failure, project personnel, and technology and system requirements have the highest likelihood to occur. Project user engagement contains risks related to user involvement, acceptance and communication within the development team. Technology failure refers to the chosen technology, programming language, and development tools, methodology, coding and testing. Project personnel encompass the risks related to experience, commitment and effectiveness of team members and project manager. Technology and system requirements consist of risks related to identifying and understanding system requirements. (Brookfield, et al. 2014, 212-214)

There are several tools and approaches to risk mitigation. Important factors are understanding what the actual risks are, which risks are perceived as more important by project managers, and how the perception of risks differs from one culture to another (Schnidt, et al. 2001, 27). Project Risk Management (PRM) can be defined as "the processes concerned with conducting risk management planning, identification, analysis, responses, and monitoring and control on a project" (Project Management Institute 2004, 127). It involves the systematic process of identifying, analysing and responding to problems that might impede the project (Tesch et al. 2007, 62). The PRM process has several proposed variations, but the general steps are: identification, analysis, response planning, tracking and control (Raz & Michael 2001, 9-10). The process is illustrated in Figure 1.

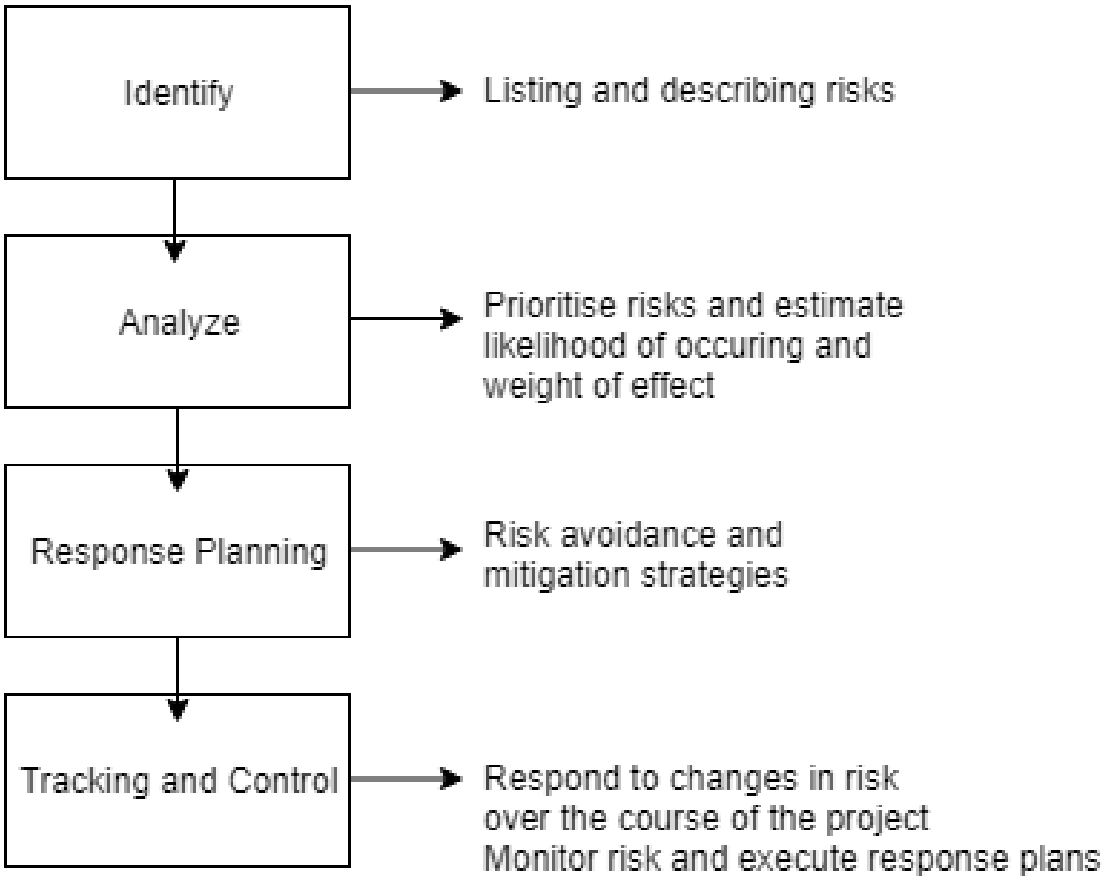

FIGURE 1. Project Risk Management Process (Tesch et al. 2007, 62)

Software development for small businesses has its own specific challenges. Small businesses are found to be affected by factors such as: a lack of inhouse IT expertise, a lack of human, financial and material resources, and attitude and willingness towards IT adoption being heavily affected by a single influential person such as the CEO (Fink 1998, 244-245). The lack of resources and IT expertise in a business means that they require software to be cheap, reliable and easy to use (Chau 1994, 300). It may be necessary to adjust the approach to project management due to company size, as there are not enough resources to cover areas such as separate quality assurance and development teams (Fayad et al. 2000, 117).

3 SOFTWARE DEVELOPMENT LIFE CYCLE

This chapter introduces the concepts of the software development life cycle and examples of specific life cycle models and their advantages and disadvantages. One of these models will be chosen and applied to this thesis project. As discussed earlier, software development faces a multitude of challenges and many projects are either left unfinished or not completed according to the original plan. This has facilitated the need for frameworks that help developers define and understand the necessary steps of software development.

3.1 Software Development Life Cycle

Software development life cycle (SDLC) is a framework that breaks down the process of software development (Techopedia.com 2018 cited 29.3.2018). This term is also known as the system development life cycle (Techopedia.com 2018, cited 1.5.2018). It covers all the stages of software from inception to deployment and maintenance (Ruparelia 2010, 8). The SDLC helps to define the structure that the development will follow and the methodology that is associated with it. The exact phases of the SDLC can vary as can the amount of importance that the development team places on each phase. There are however key phases that are common to most software projects. These phases can be called Requirements Analysis, System Design, Implementation, Integration and Deployment, and Operation and Maintenance (Maciaszek 2007, 26,30).

Requirements analysis can be defined as determining and specifying customer's requirements for specific software. A requirement can be defined as an expected functionality or constraint upon the software. Functionality and constraint statements describe how the system should behave with users and what restrictions should apply to the system. (Maciaszek 2007, 30-31) Requirements are a way of specifying what the software's business impact will be, what the customer wants and how end users will interact with the software. The goal of requirements analysis is to ensure that developers have a clear understanding of what they should be building before they start development. (Pressman & Maxim 2015, 132-133)

Requirements analysis can be divided into two distinct parts; requirements determination and requirements specification. The purpose of requirement determination is to determine, analyse and

negotiate requirements with users by gathering information from them. Tools such as questionnaires, interviews and prototypes can be used to gather and clarify customer requirements. (Maciaszek 2007, 31) Determining user requirements can be challenging. Users may not fully understand their needs or expectations for the software and their needs can even change over the course of the project. Lacking or misunderstood user requirements can lead the project astray from the beginning and lead to a software product that does not truly match the user needs. (Pressman & Maxim 2015, 131-133)

When the requirements have been determined, requirement specification aims to give a more detailed explanation through modelling the requirements. Functional requirements describe what functions the system should be able to accomplish. Other requirements such as performance, usability, security and legal may also be described. The specification models should generally be independent of hardware and software considerations, but it can sometimes be a necessary inclusion, should the customers require the use of specific technologies. Keeping this phase from getting too technical makes it easier for customers to understand, which makes communication and requirements gathering easier. Requirements analysis produces a requirements document, which is initially mainly narrative documentation of the determined requirements with informal diagrams and tables. The document is essentially replaced by a specifications document after the modelling work of the requirements specification. The specification document lists all the requirements that the software must meet. (Maciaszek 2007, 30-32)

In system design the logical model of the system is modified until it represents what the new system will do and how the system will accomplish its objectives will be determined (Marakas & O'Brien 2011, 492). System design encompasses the structure of the software and details of internals and interaction of system components. System design is derived from the findings of the requirement analysis. Design includes software and hardware considerations associated with the development, unlike requirements analysis where they are often ignored. (Maciaszek 2007, 32)

System design can be divided into architectural design and detailed design. Architectural design is a description of the system in terms of its components. It is concerned with resolving user interface and database issues as well ensuring those processes interact correctly. Architectural design has a substantial impact to the long-term success of the system. When done well, it produces an understandable, maintainable and extensible system. This means that the system and how it operates should be explainable to users without technical expertise, the design should consider

the effect of maintenance and downtime and that the system should be able to grow as the user base or the needs of the customer grow. (Maciaszek 2007, 32-33)

Detailed design describes how the software components function internally. The design of the components is subject to constraints caused by technology associated with the development. The user interface might have to conform to design that supports Internet browsers in the case of web-applications, for example. (Maciaszek 2007, 33) System design can be considered to deliver three major products; user interface, data structure and process design (Marakas & O'Brien 2011, 492).

Implementation involves the acquisition and installation of hardware and software necessary for development, coding the software under development, testing the developed software and training end-users in using the software (Marakas & O'Brien 2011, 503,506). Based on the findings on previous SLDC phases, the software and hardware requirements that the software under development requires should be clear. These can mean both development tools and the platform the software will be deployed to. The implementation team will set to programming the software according to the design documentation refined in the system design phase. The software can go through many changes in this phase, as the development team gets feedback from the customers. (Maciaszek 2007, 34)

The aim of implementation is to create software of quality. Software quality can be defined as the degree to which a system, component, or process meets specified requirements and customer and user needs and expectations. Methods for ensuring software quality can be split into Software Quality Assurance (SQA) and Software Quality Control (SQC). (Software Testing Fundamentals 2018a, cited 30.3.2018) SQA contains activities ensuring quality in software engineering processes, which will then result in the creation of quality software. SQA is a preventative measure that is looking to prevent problems from arising in the first place. (Software Testing Fundamentals 2018b, cited 30.3.2018) SQC is a set of activities that identify defects in software products during and after development. SQC is a responsive measure to problems that have already occurred. (Software Testing Fundamentals 2018c, cited 30.3.2018) While some models include testing as a phase in the SDLC, others run it as a separate activity parallel to the SDLC spanning the whole lifecycle, rather than as one of the phases (Maciaszek 2007, 35). A test can be defined as an act of using something to find out if it is working correctly or how effective it is (Cambridge English Dictionary 2018, cited 30.3.2018). Testing is one of the SQA processes (Software Testing Fundamentals 2018b, cited 30.3.2018).

Integration is the process of putting the software modules together. In large systems integration can take more time and effort than any earlier SDLC phase and should be planned for from the start of development. Modules that need to be implemented individually should be identified during system analysis, designed in more detail during architectural design and implemented in a sequence that allows for as smooth integration as possible. The major challenge of integration comes from the dependencies between software modules, where a module will not function properly without another, or changes in one will force changes in another. When modules are too dependent on one another, it can create a domino effect where changes in one module force sweeping changes across the system. Deployment signifies the hand over point in development when the software is moved from the developers' local platform to the customer's platform. (Maciaszek 2007, 34)

Operation phase is when the software has been delivered to the customer and becomes a part of their operations. If the software is replacing a previous solution, software or not, the process of switching to the new software is often gradual and run parallel with the old method as a failsafe. (Maciaszek 2007, 35) In this phase the software is in operation and can receive enhancements and changes. The software is being monitored and its performance in relation to customer requirements is being evaluated. The evaluations will determine how the software needs to be changed, or how it can be improved and expanded upon. The operation and maintenance phase continue until the software is replaced or phased out. (Radack 2003, 4)

According to estimates 75 percent of lifecycle time is spent on maintenance post deployment. Maintenance can be divided into three stages: housekeeping, adaptive maintenance and perfective maintenance. Housekeeping involves routine day-to-day activities to keep the software operational. Adaptive maintenance involves monitoring the software and adapting and adjusting its functionality to meet customer demands. Perfective maintenance involves redesigning or modifying the software when new or significantly changed requirements appear. (Maciaszek 2007, 35)

3.2 Software Development Life Cycle Models

The SDLC identifies stages and tasks necessary for software development. The SDLC is however more of a model of what should be done rather than how it should be done. There are several

SDLC models that take a different approach to the life cycle. The following section will introduce some of these models and consider which is suitable for this thesis.

3.2.1 Waterfall model

Waterfall is the oldest SDLC model. It takes a linear and sequential approach to software development, where each SDLC phase is completed in their entirety until moving on to the next. Waterfall can be a suitable model choice in situations where well-defined enhancements are made to software. It can also be used in new development projects, but only requirements are fully understood from the start and are not expected to change. (Pressman & Maxim 2015, 41) The process of the Waterfall model is shown in Figure 2.

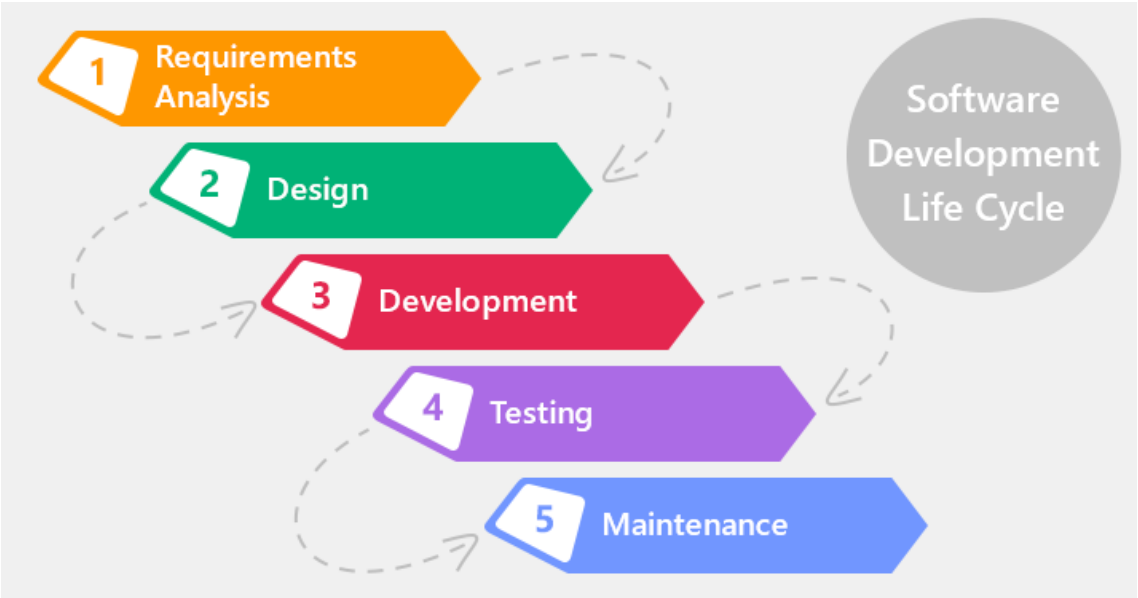

FIGURE 2. Waterfall model (Gordiyenko 2014, cited 7.4.2018)

Waterfall is often found unsuitable for modern development software development. The situation where requirements are so clear and stable that no changes are expected is extremely rare. Customers often have difficulty stating what their requirements are and they are found over the course of the development after reviewing design models and prototypes. The sequential model makes it hard to make changes based on customer feedback, since phases are completed one at a time and in their entirety. The customer will also not see a working version of the product for a long time, since implementation takes place later. (Pressman & Maxim 2015, 42-43) The assumptions that the waterfall model makes are that: the requirements are known before

implementation, the requirements have no unresolved, high-risk implications, the requirements will not change much, the requirements meet the expectations of customers, users, and developers, and that the schedule allows for sequential SDLC process (Boehm 2000, 7).

3.2.2 Prototyping

Prototyping is an evolutionary SDLC model that develops software through multiple iterations. It is found that requirements often change or are discovered during development. The iterative approach makes it easier for the development to adapt when this occurs. Prototyping is can be used when requirements are left unclear. Developers run quickly through the SDLC phases with focus on user interface and aspects of the software visible to end users to get feedback from customers. They develop a simple prototype and see if the solution matches customer needs. The prototype is improved upon by each iteration as the requirements become clearer and eventually evolves into a fleshed-out product. Though prototyping can be a SDLC model of its own, it is often used as a requirements identification technique as a part of other frameworks. (Pressman & Maxim 2015, 45)

Prototyping is not without its issues. Sometimes customers can become too enamoured with the prototype, mistaking the surface level functionality for robust design. They may insist on cutting development time that would be used to ensure the systems reliability and maintainability in the long run. It is important to communicate what the prototype lacks and what problems taking shortcuts can cause. The developers should also take care to evaluate their chosen tools, platform and programming solutions as the prototyping process advances. Time constraints to provide a prototype quickly can lead to unsuitable choices for the software in the long run. The key to a successful prototyping model is to make it accepted among customers and developers that the prototypes are an incomplete, disposable solution that needs more work until a quality product is made. (Pressman & Maxim 2015, 46-47)

3.2.3 Spiral model

The spiral model is an evolutionary SDLC model that aims to combine the sequential approach of waterfall with prototyping from iterative models (Pressman & Maxim 2015, 47). The aim is to develop software through iterations that start small and simple and grow more complex with each

new cycle. The spiral model aims to take a risk-driven approach to software development instead of a documentation or code-driven approach. (Boehm 1988, 61) The model can be divided into four quadrants that each iteration cycle passes through. These are: determine objectives, identify and resolve risks, development and test, and planning the next iteration. (Ruparelia 2010, 11) The phases and detailed activities within them are shown in Figure 3.

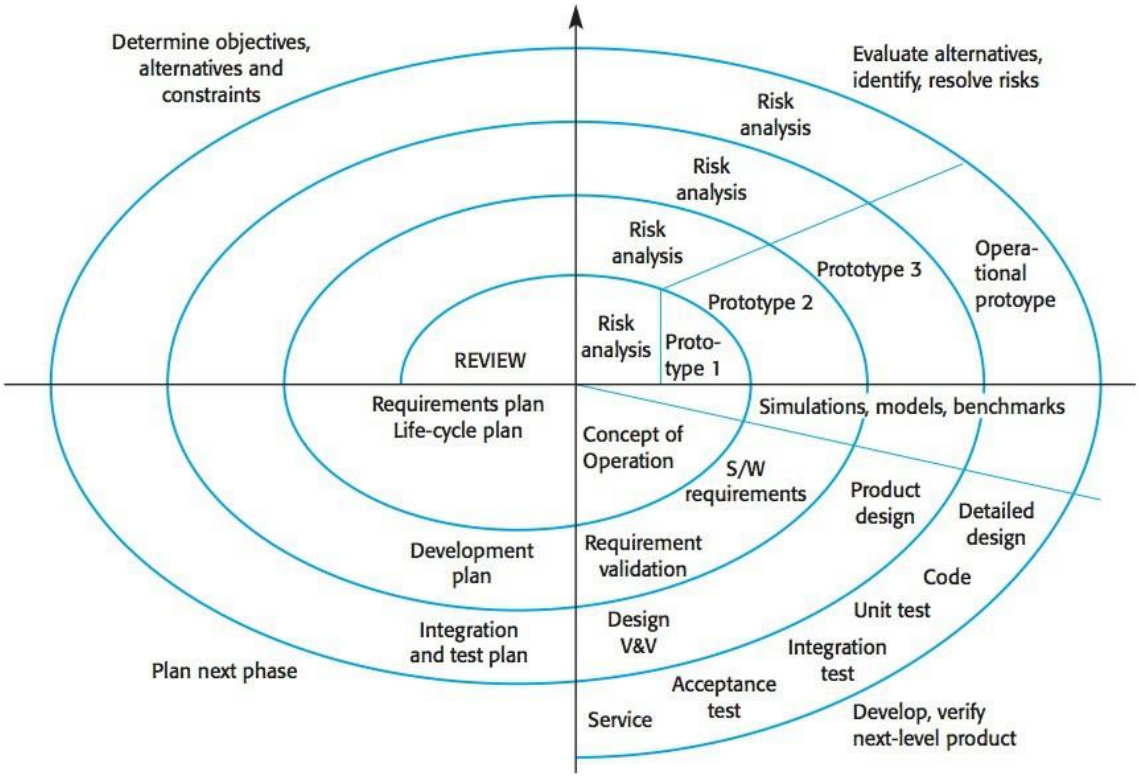

FIGURE 3. Spiral Model (Boehm 1988, 64)

Each cycle of the model begins with determining the objectives, alternatives and constraints of the portion of software being developed. The content of the determining phase will vary based on what stage of development the current cycle is on. At the start of development this will involve determining requirements and constraints such as customer needs, scheduling, cost and budget. The next step is to identify and categorise risks into performance or user interface related risks and development risks. If performance related risks are dominant, development will go through iterative steps, where prototypes are developed, and risks are resolved. Feedback received from the prototype is used as a basis for the next cycle where the steps are again repeated. When the prototyping leads into resolving the performance and user interface related risks and the prototype is robust enough to serve as a basis for further development, the spiral continues through the remaining SDLC phases. (Boehm 1988, 64-65)

The spiral model is a good approach to development of large scale systems and software. It allows developers and customers to understand and manage risk at all stages of the project. It allows for more adaptability than the waterfall model and better risk management than relying on pure prototyping. (Pressman & Maxim 2015, 49) Due to the heavy investment in risk management however, it is difficult to apply in smaller projects and is very much reliant on risk management expertise. Projects that do not have the time or personnel to reach the necessary risk management standards will likely find the model unsuitable. (Boehm 1988, 69-71)

3.2.4 Agile Scrum

Agile is a term used to describe several software development models that share the same set of values and basic principles. The Agile models are a response to traditional more documentation driven approaches to software development by developers that found them frustrating, restrictive or not accurately reflecting the reality of software development projects. Agile considers important the ability to respond to change, improving communication between stakeholders, rapid delivery of operational software and involving the customer as part of the development process as much as possible. (Pressman & Maxim 2015, 66-68)

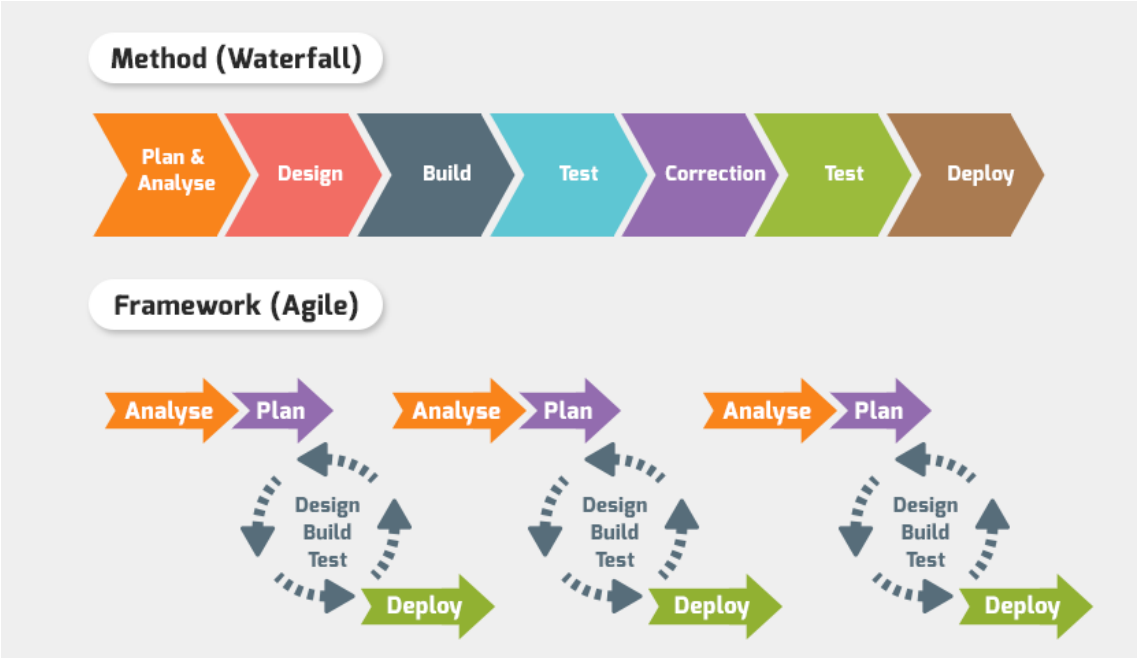

FIGURE 4. Agile Framework (QuickScrum 2016, cited 7.4.2018)

Figure 4 compares the Agile and Waterfall models. As stated, there are several models that fall under the Agile term. A recent survey shows that 94 percent of their respondents are using Scrum in their Agile practices (Scrum Alliance 2017, 9). There are other popular Agile models such as Extreme Programming (XP) and Kanban, but this thesis will focus on Scrum.

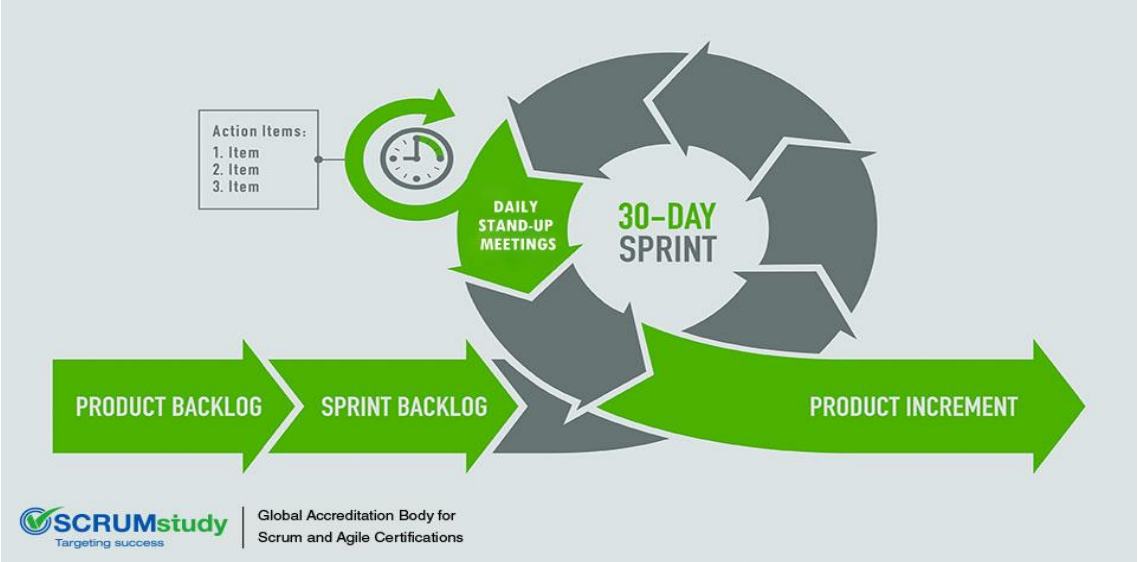

FIGURE 5. Scrum Framework (SCRUMStudy 2017, cited 8.4.2018)

Development in Scrum is divided into cycles as illustrated in Figure 5. The cycle begins with a stakeholder meeting where the general project plan is formed, and a product backlog is created based on requirements analysis. The backlog is a list of development tasks that relate to the current requirements that have been identified. The backlog is updated as the development progresses and new requirements are identified, or they change. The product backlog is divided into smaller sprint backlogs. A sprint backlog consists of tasks that are to be completed during a sprint. A sprint represents a unit of development time that has certain tasks assigned to it according to the sprint backlog. A typical sprint period is 30 days, but this can be subject to change. There are daily Scrum meetings where each member of the team will answer three key questions: what you did since the last meeting, what obstacles are you encountering, and what do you plan to accomplish by the next meeting. The meeting lasts about 15 minutes. Towards the end of the sprint a review meeting is held where the deliverables of the sprint will be examined by stakeholders. A sprint cycle ends with a discussion of how performance could be improved, and the next cycle begins. (SCRUMStudy 2017, cited 8.4.2018)

A properly implemented Scrum can offer several benefits. It enables projects with unclear or hard to quantify requirements to be successfully developed. Developments can be quickly coded and tested, and mistakes can be easily corrected. Dealing with changes becomes easier due to short sprints and continuous feedback. The daily meetings allow for early identification and resolving of problems. (Adell 2013, cited 27.4.2018) Scrum also decreases the time it takes for software to become available on consumer markets. This is achieved by reducing the delay of implementation phase by cutting the waterfall documentation phase, prioritising important requirements and leaving the less important ones to post launch development and producing an operational product at the end of each sprint. (Layton 2018, cited 8.4.2018)

There are Scrum related issues however. Scrum is one of the leading causes of feature and scope creep in projects, because non-definite deadlines and heavy customer involvement during development often tempt demands of increased functionality (Adell 2013, cited 27.4.2018). It is found that experienced and committed development team members are required, as the pace of the project can be very demanding. Members leaving, or underperforming can quickly lead to delays and other issues. As such, the success in Scrum projects is very dependent on the expertise of personnel involved. (Uhlig 2018, cited 8.4.2018)

4 DEVELOPMENT PROCESS

This chapter will illustrate the process of system development done during the thesis. One of the SDLC models introduced previously will be chosen for use in the thesis. The development will then be carried out according to the steps and methodology suggested by the chosen model.

4.1 Thesis SDLC Model

When choosing a SDLC model for this thesis, there are two different scenarios to consider. The first is a scenario where the student café had stayed as the client for the thesis. The second is the current situation, where the café is only acting as the basis for the system's requirements but will not affect the development beyond that. The presence of a client has a significant effect on the suitability of a model.

In a scenario where a client was present, the chosen model for the thesis would be a modified agile Scrum. Since in this scenario the system would be in daily operation, it is necessary to be able to receive customer feedback and alter the system accordingly. Scrum would allow for this to happen and get prototypes out quickly, so feedback can be gotten early on and regularly. Though spiral model and prototyping would allow for this as well, spiral model with its heavy investment in risk management is somewhat too heavy for a small-scale project that is under time pressure and prototyping serves better as a technique used within an another SLDC model rather than completely as its own model. The scrum model would however require some modification, especially in terms of the distinct roles assigned and the daily meetings between developers. The thesis has only a single developer and thus needs to carry out all the tasks that would normally be divided between members.

In the current scenario of the thesis where no client is present, the chosen model for the thesis is waterfall. The system being developed is based on the requirements of the café, but they are not involved with the development beyond that. This means that the requirements are static and are not subject to change due to client feedback. Since the biggest issue with waterfall is uncertainty regarding requirements and ability to adapt to changes, this eliminates the problem. Waterfall

allows for a clearly defined and planned development process with each stage producing specific deliverables. This helps in understanding and scheduling the development process.
The lack of client does make some phases of the SDLC redundant however. Since the software will neither have a point where it is handed over to the client nor will it be in daily use and require maintenance, the SLDC phases of deployment and operation and maintenance will be left out. The suggested SDLC of the thesis would be: Requirement Analysis, System Design, Implementation, and Integration.

4.2 Requirements Analysis

The development began with meetings with the café's representative. The topics of conversation were the current system used by the café and what they expect the new system to do. The current system used by the café involves the customers buying a cardboard card with limited credit that can be spent on coffee purchases. Each time the customer wishes to buy a cup of coffee, one of credit is expended. This continues until the card runs out of credit, upon which time a new card needs to be purchased. The new system would not need to track purchases as the café already had a system that does so.

The café wanted to replace the cardboard card with the SAMOK student card. The student card should act as an identifier for each customer. The exact method of involving the student card with the system was left up to the developer, but the possibility of using a card reader to achieve this was discussed. The café has one computer running on the Windows 10 operating system and this computer was considered as the platform to host the system. Deploying the system online was initially considered but was eventually left out as there was not sufficient reason to do so. The reason driving online deployment was the possibility of the system interacting with existing databases containing student information for increased automation but achieving this would have required co-operation with the school IT administration and caused the scope and security considerations of the work to blow out of proportion. Based on the notes taken during the meetings it became possible to determine requirements for the new system.

After the meetings with the café, the development was put on hold for several years due to personal reasons on the developer's side. During this time the administration of the client had changed several times over and the original plan of delivering the system for the café could not be continued.

The absence of a client made it necessary to make changes to the original approach. The system would still be based on the requirements of the café, but the priority and feasibility of some requirements changed. The system is meant to be more of a proof-of-concept model rather than a product meant for day-to-day operation. This means that the importance of non-functional requirements such as security, maintainability and reliability is lower than it would have been originally. The implementation of a card reader to interact with the student card was also left out due to budgetary limitations.

TABLE 2. User requirements

Requirement	User type
Login	Staff/Admin
Modify own account	Staff/Admin
View customer list	Staff/Admin
Manage customer account	Staff/Admin
Scan for customer account	Staff/Admin
Create new customer account	Staff/Admin
Add credit to customer	Staff/Admin
Remove credit from customer	Staff/Admin
Create new staff account	Admin
View staff list	Admin
Modify staff account	Admin
View available credit	Customer
Buy credit	Customer
Buy coffee	Customer

Table 2 lists the functional requirements of the system categorised by user type. Functional requirements refer to what the system should do or what users should be able to do with it. Three distinct user groups were identified during this phase. Staff represents the café staff and they are the main operators of the system. Admin in this case represents a staff member that has been entrusted with added functionality and influence in the system. Customers represent the customers of the café that will not directly operate the system but will have their data stored and managed and will make requests to the staff. The staff needs to be able to login to the system, modify their user account information, view and manage existing customer accounts, scan for customer account,

create a new customer account, add coffee credit to a customer account, and remove coffee credit from an account. The admin inherits all requirements from staff but has the added requirements of modifying existing staff account information and creating a new staff account. The customer needs be able to view their existing credit, have new coffee credit added to their account, and spend their credit.

The choice of tools and programming languages was left up to the developer. The choices were eventually narrowed to HTML, CSS, JavaScript, PHP, MySQL and Apache HTTP server. HTML and CSS were chosen to create the look of the user interface. PHP and JavaScript allow for the creation of dynamic web pages. PHP is also used in conjunction with MySQL to interact with the database. A web application needs a host platform to run on and the Apache server allows this. These tools allow for the creation of a light-weight system with low resource requirements.

4.3 System Design

The scope and structure of the system was determined by the needs of the café and the technology choice of the developer. From the point of view of the café the system needed to be lightweight, locally hosted, easy to use, and low budget and resource cost. There was no guarantee that the café staff had the technical expertise to operate a complicated system. The development budget was low even with the client and became more restricted after the co-operation ended. Additionally, the choice to move towards a functional model rather than a full system meant the system should be expandable, should development ever continue towards full implementation.

The hardware components of the system consist of the host computer and in a full system, a card reader. In this version of the system scanning for customer accounts was designed to work by a staff member inputting the student card number manually. It would also be necessary to set up an email server for delivering and recovering passwords for the café members. In this version, the password management was left to the administrator user group. The software components of the system consist of the code base of the system itself, a web browser, Apache HTTP Server and the database. The database of the system has only two tables in it: one for staff accounts and one for customer accounts (Figure 6). The customer_card_number is used to identify customer accounts and customer_credits represents the amount of credits the customer can spend to buy coffee. One

credit equals one cup of coffee and keeping track of real life pricing is not within the scope of this system.

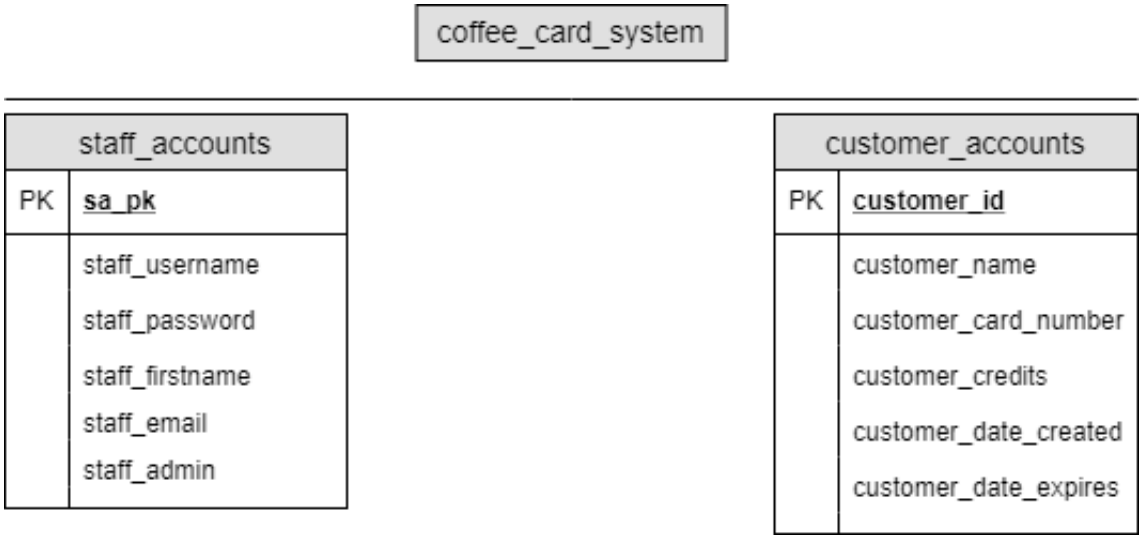

FIGURE 6. System database

The layout of the system is illustrated as a site map in Figure 7. The point of entry into the system is the login page. After a successful login the user is free to navigate freely between the other pages. The user is automatically taken to the point of sale page after login, which contains the customer scanner. Transaction management also occurs on this page. The navigation is implemented with a navigation bar. The staff accounts page is restricted to administrator users only.

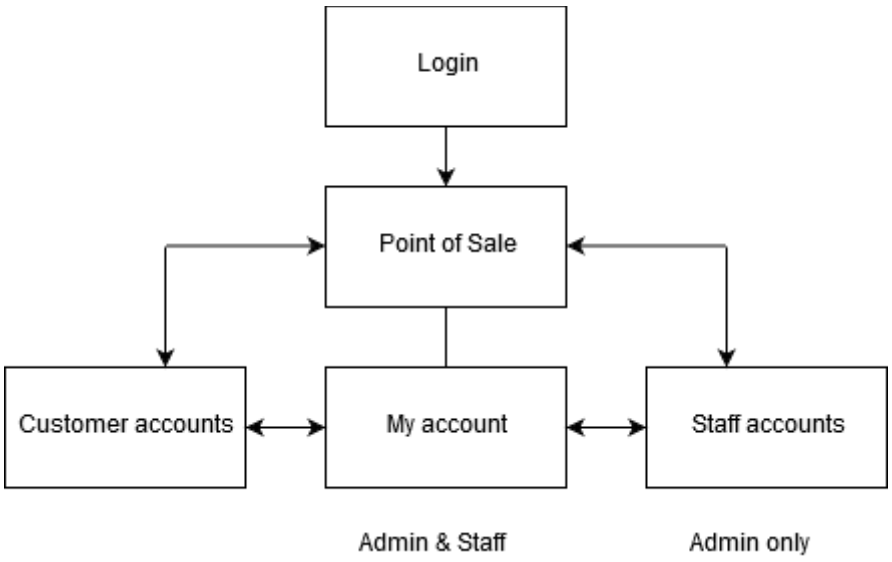

FIGURE 7. Site map

4.4 Implementation

The first components of the system to be implemented were the login page, transition to the piece of sale page and the staff portion of the database. The login system allows splitting the staff into administrators and non-administrator users. The difference between the two groups is giving administrators access to create and manage staff accounts. The look of the login page which is shown in Figure 8 was achieved with HTML and CSS and PHP was used to connect to and retrieve information from the database.

FIGURE 8. Login page

The login page is used to secure the system from unauthorised access. The staff are given a username and password to access the system. On successful login the system sends the user to the point of sale page (Figure 9). On unsuccessful login attempt the system stays on the login page.

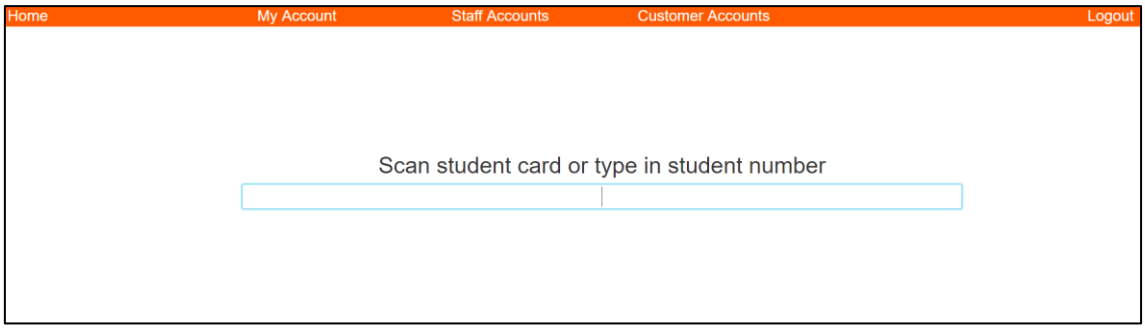

FIGURE 9. Point of Sale page

The top of each page past the login has a navigation bar. This allows the user to access different system features. The staff accounts link is only visible to administrators. The point of sale page has the scanner for the student number. On a fully functional version of the system this would work in conjunction with a card reader. The system will take the input from the scanner and compare it to the database. Scanning the number has three possible outcomes: 1. the account does not exist, 2. the account exists but has no credit, or 3. the account exists and has credit. Outcome 1 is illustrated in Figure 10.

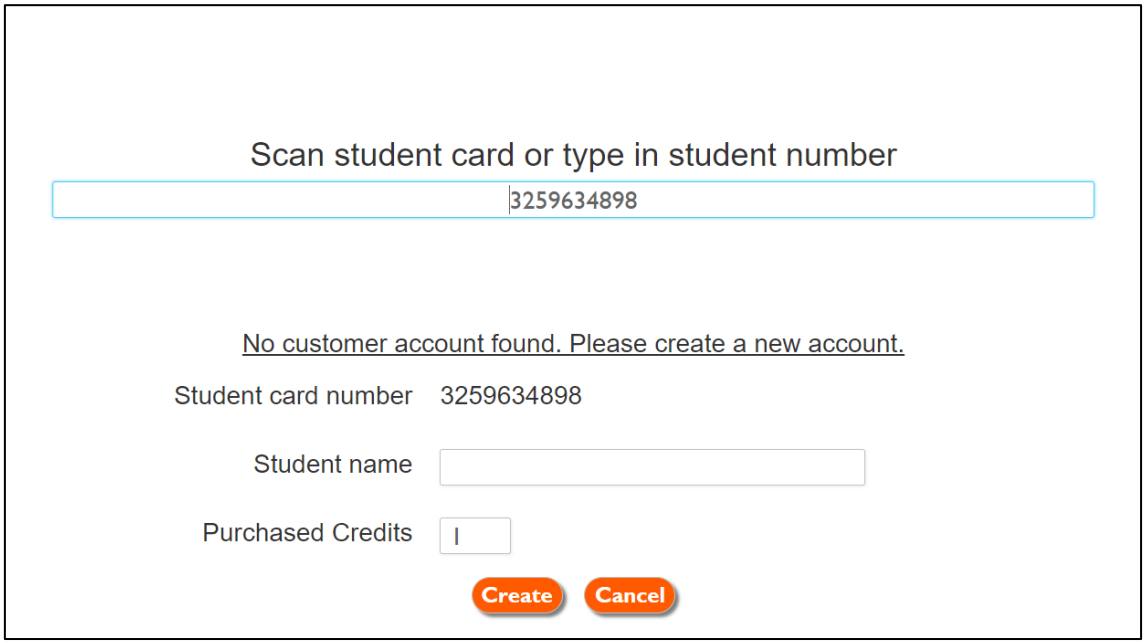

FIGURE 10. No customer account found

If the given student number does not match an entry in the database, the system prompts the staff to create a new customer account. The system asks for a student name and amount of credits they wish to purchase, if any. Clicking cancel will interrupt the process and remove the account creation

prompt. The system also enters the date of creation and date of expiry into the database. The date of expiry is set to be five years after the account has been created.

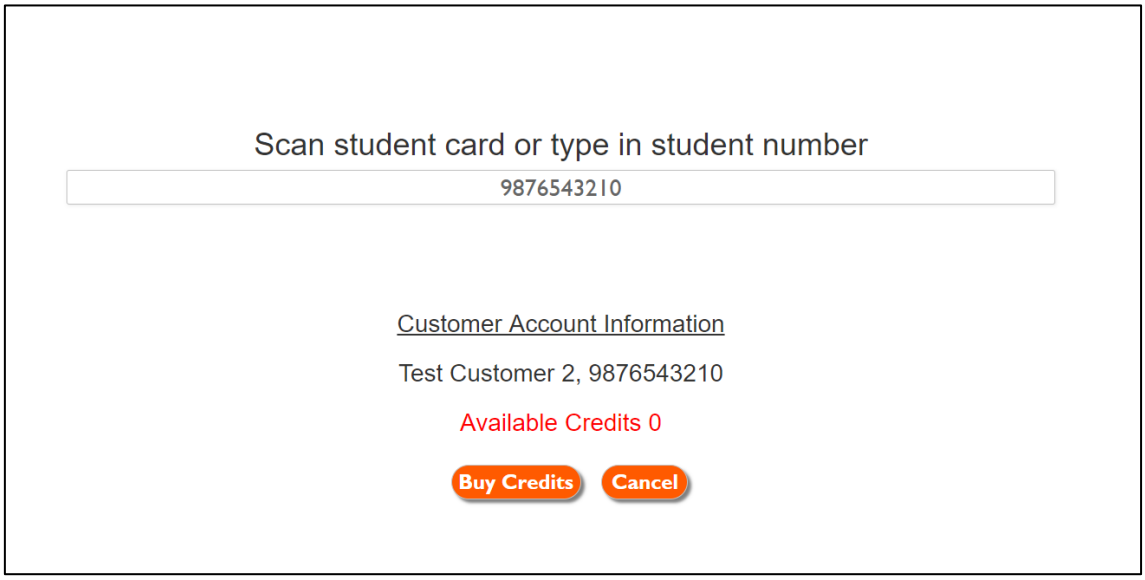

FIGURE 11. Account with no credit found

Outcome 2 is shown in Figure 11. If the system finds a matching entry in the database but the account has no credit, it prompts the customer to purchase some. Cancelling is made possible, in case this was just a check on available credits or the customer has second thoughts. Clicking 'Buy credits' will take the user to the purchase page shown on Figure 12.

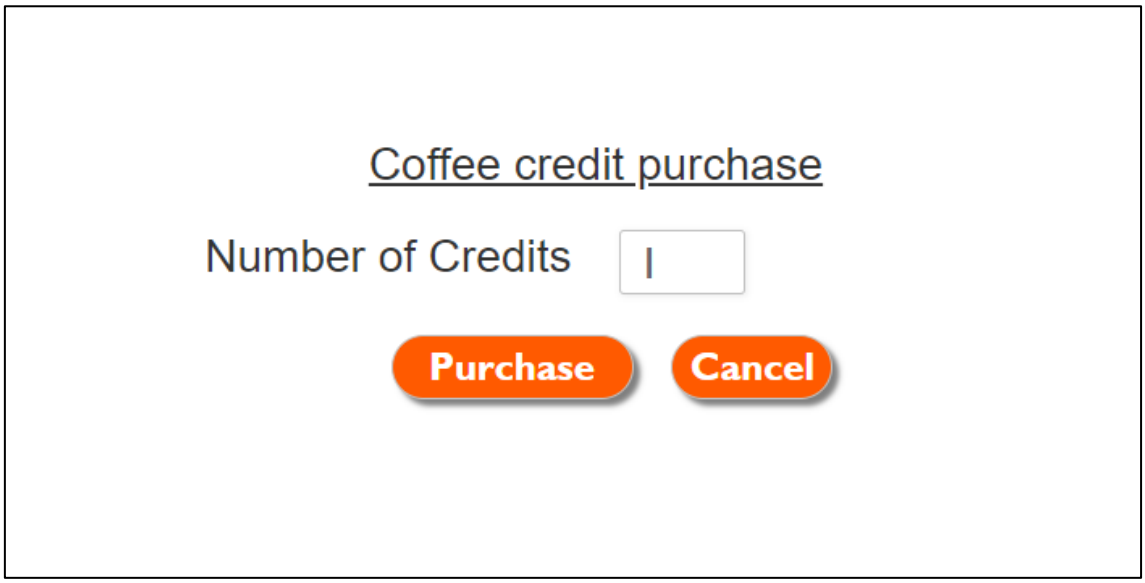

FIGURE 12. Purchase credit

The user is asked to enter the amount of credit they wish to purchase or cancel the order. Either choice will take the user back to the point of sale page.

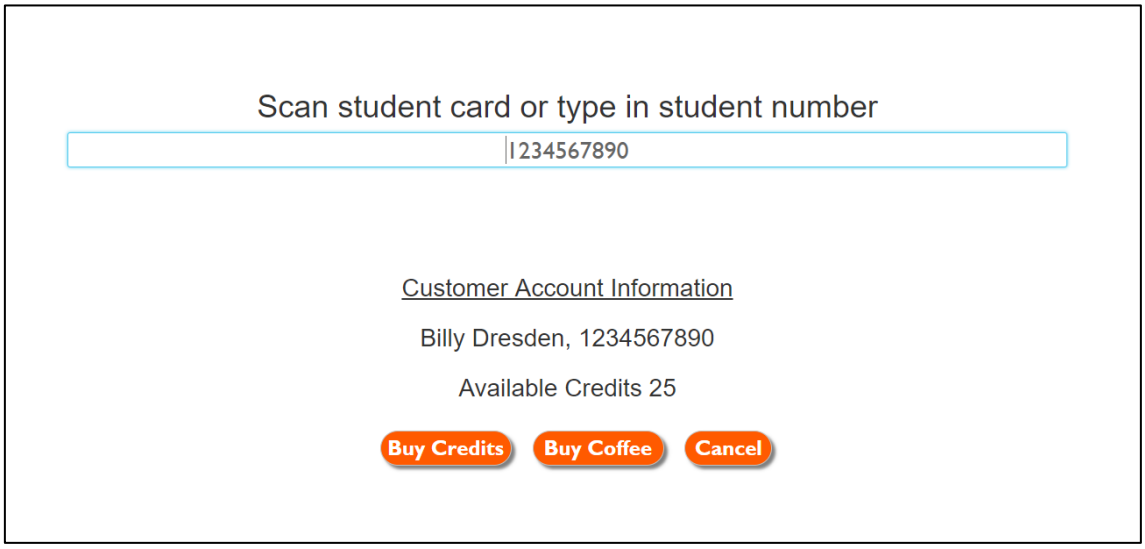

FIGURE 13. Account with credit found

Outcome 3 is illustrated in Figure 13. When the number matches an existing account that has credit, the system shows the currently available credits and allows to purchase more credit, purchase coffee, or cancel. Purchasing credit is identical to Figure 12 and purchasing coffee reduces the available credit by one.

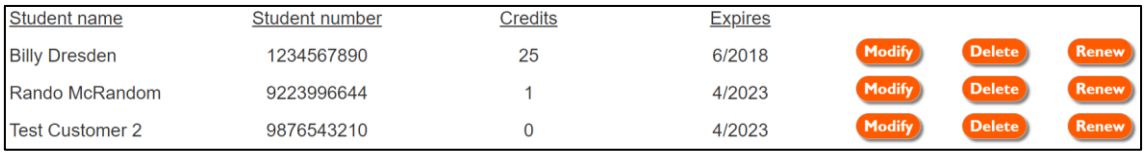

FIGURE 14. Customer accounts

Figure 14 shows a list of existing customer accounts. This is accessed via 'Customer accounts' in the navigation bar. The feature is available to both staff and admin users. They can delete an account, modify account information or renew an account. Renewing will set the expiration date to current month but five years from now. Choosing to modify an account will lead to the page seen in Figure 15.

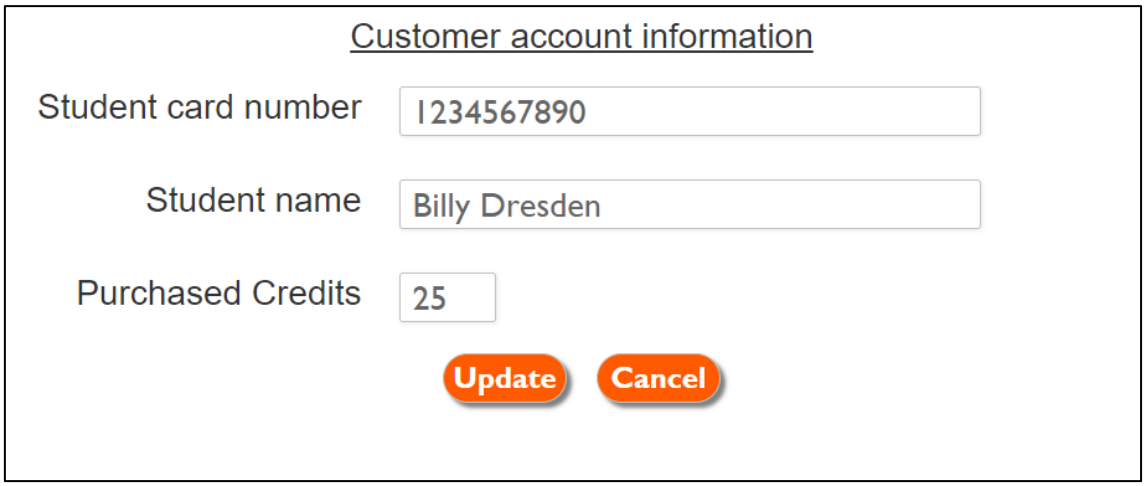

FIGURE 15. Modify customer

Customer information should be modifiable just in case. If a customer somehow loses their card and wants to link it to their old account, changing the card number should allow for this. The staff can also undo accidental purchases. Clicking cancel takes the user back to the customer list.

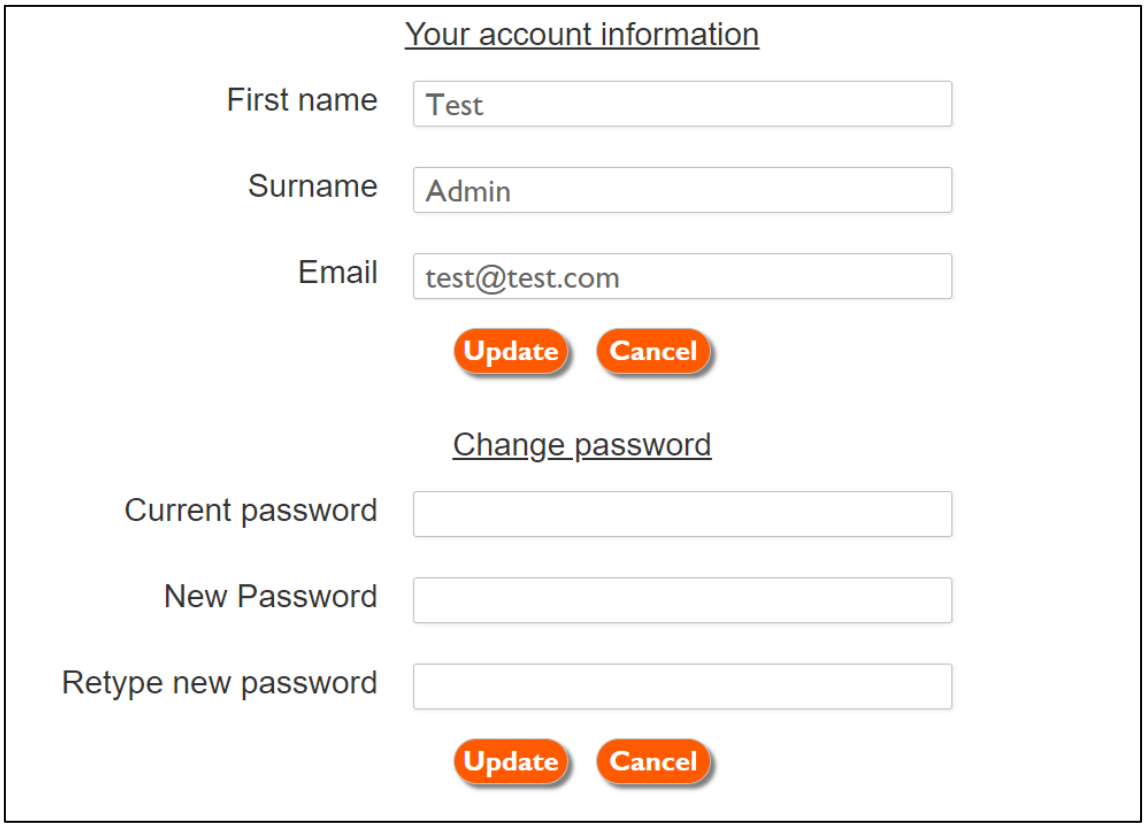

FIGURE 16. My account

Figure 16 shows the 'My account' feature from the navigation bar. The page has forms for personal information and password. Changing the password requires entering the current password and for

the two fields for the new password to match before changes are made. Clicking cancel takes the user back to the point of sale page.

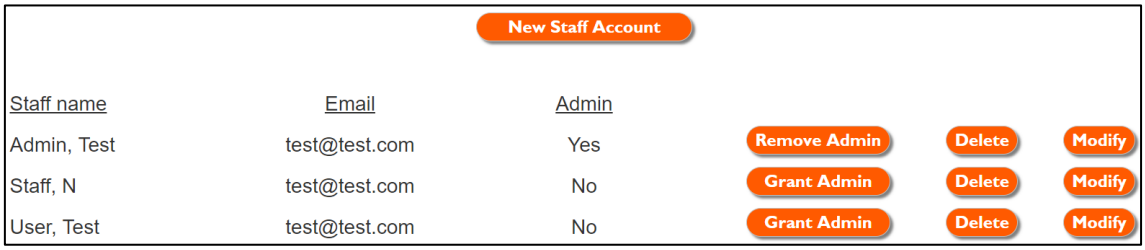

FIGURE 17. Staff account control

Figure 17 shows the list of existing staff accounts. This is only accessible to administrators from the navigation bar. The admins can grant admin status or take it away, delete staff accounts or modify the password of an account. The password modification would not be present in a full system but would be replaced with password recovery through email. Implementing a functional email server for the thesis was not possible however. The administrator can also create a new staff account. The creation page is shown in Figure 18.

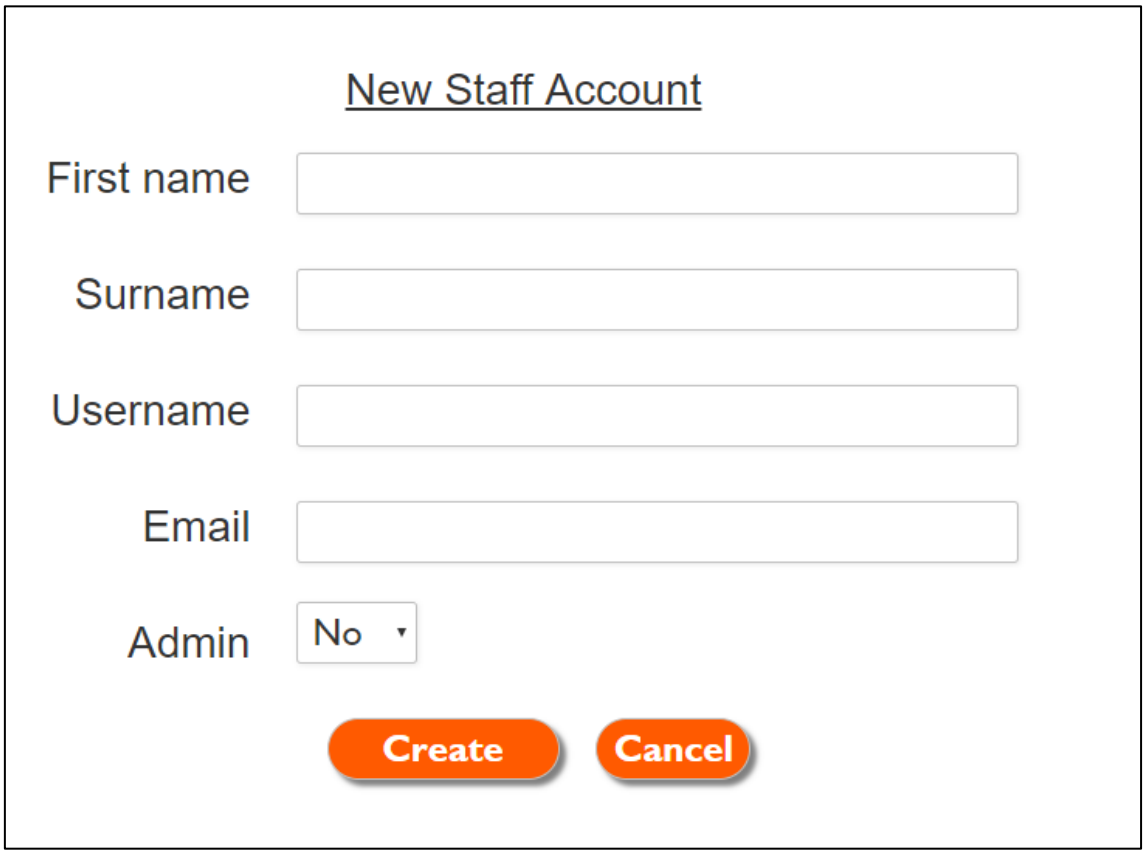

FIGURE 18. Staff account creation

The creation of a new staff account takes user input for all fields but the password. For testing and demonstration purposes the new password has been set to a constant value, but in the full system the first password would be random and distributed through email. The users would then be asked to change their password after their first login. Clicking cancel takes the user back to the staff list. The system does not send any data until all fields have received input.

5 DISCUSSION AND CONCLUSIONS

The main objective of the thesis was originally to deliver a customer account control system for a student café. Delays due to personal reasons of the developer caused the co-operation with the café to end. Once work was resumed the focus of the development shifted to providing a functional model of a system rather than one meant for daily operations, which caused changes to the requirements of the system. The development was carried out following the Waterfall SDLC model. The final system matches the adjusted requirements. Despite several unavoidable real-life issues unrelated to, but affecting the development, the thesis was able to adapt and proceed to its conclusion. The work done during the thesis was both familiar and new and allowed me to learn more about tools I was already familiar with and begin learning new ones.

The time spent working for the student café was short, but I could identify challenges that development for the café or similar organisations could face. The biggest influencing factor is the regularly changing staff and administration at OTRO. OTRO has its members chosen through elections, which gives a limited period for the project to be carried out under a single administration. Projects that cannot be completed within that time risk getting lost in the shuffle during the changing of administration. It is important to make sure that details about the project are passed on to the next administration and that communication between them and the developer are established. There is however a risk that new the administration will not be as interested in the project and will see fit to discontinue it. Other than the time pressure caused by elections, development for OTRO will also have prepare for low budget and the lack of technical expertise among the organisation.

The end of co-operation with the café had a significant effect on the thesis. It became necessary to re-evaluate priorities for both the written content and the system. The overall focus of the thesis shifted from delivering a fully functional version of the system for the café's daily operations to examining the challenges of software development for small enterprises through the case of the café. The development became more focused on delivering a functional prototype that would serve as a model solution to match the needs of the café but did not have as much polish or focus on non-functional requirements as it originally did. One major component of the originally planned system that was left out was the card reader. The idea originally came from the café with the expectation that they would cover the costs of getting the hardware. Since this was no longer the case, the card reader was excluded due to limited budget. The cost of a piece of hardware that

would not see any use afterwards was too much to justify. The SDLC model was also affected as the original plan was to follow the Scrum model and carry the work out in sprints, but the lack of client feedback and expected changes made me choose Waterfall.

The experience following the Waterfall model was a mixed one. The heavy design phase did allow for smooth implementation, but I certainly felt constrained many times during development having to adhere to the strict sequence of phases. This made the work often feel frustrating and made me understand the sentiments behind the Agile manifesto that caused its creators to explore alternatives in the first place. Overall, I think Waterfall was a suitable method for the project, but not necessarily for me as a developer. The one-man development team also meant that there were no other teams having to wait for previous sequences to be finished, which often bottlenecks Waterfall projects. If I were to do the project again however, I would look towards Agile models.

There were both familiar and new tools used during the thesis. I had worked previously with HTML, CSS, SQL and PHP including during my practical training, so the choices felt natural. Using JavaScript was a first for me and something I want to learn more in the future. When creating the look of the site with HTML and CSS I used a tool called Bootstrap for the first time. Bootstrap is an open source HTML, CSS and JavaScript framework that is used to help in designing web applications. Getting introduced to JavaScript and Bootstrap were two important lessons going forward. I will definitely continue to work with both in the future. Though the scope of the system ended up being narrower than originally planned, it was still more complex than any project I had previously worked on.

I would overall consider the thesis a success, but it did have its challenges. The biggest challenge came in terms of scheduling and delays. The factors causing the problems were outside of my ability to control and they have prolonged the completion of the work immensely, being also responsible for the co-operation with the café ending. That caused further work with the goals of the thesis to having to be adjusted so that work could continue. Despite the difficulties however, the thesis was able to adapt and continue. After the issues delaying the work were resolved the thesis progressed at a good pace, reaching its conclusion in roughly a period of three months. The development process went smoothly without any major issues with the tools chosen. The choice of Waterfall did not impede the development itself, but it did frustrate the developer somewhat.
The system created for the thesis could be expanded or adjusted in several ways. There is a lack of polish and usability features that the system would need before being ready for daily operations.

The system could be subjected to more rigorous testing especially in terms of its reliability. Functionality that could be added to the system without significantly altering its main purpose could include tracking for sales. The system could track and allow for reporting of which products are popular or which time of the day does the café see most traffic. The system could also be expanded into tracking inventory by adding product types to act as identifiers. The current system is designed for local deployment and if further development were to aim for online deployment, there would be a need to enhance the security of the system.

The work done during the thesis allowed me to improve as a developer. I gained insight into the process of software development and the difficulties that are involved. I was also able to gain experience working with tools I was already familiar with, but on a more complex project than ever before. I was able to also introduce myself to tools I was previously unfamiliar with. Due to the problem that appeared during the thesis I understand the importance of communication and client importance more so than before. If I were to carry out a similar project in the future, there are a couple of things I would like to focus on. I would like there to be more client involvement this time around and I would also like to carry out the development following Scrum or other Agile model due to them being so different to Waterfall and popularly used methods.

REFERENCES

Aaker, D. A. & David, M. 2010. Strategic Market Management: Global Perspectives. Chichester: John Wiley & Sons.

Adell, L. 2013. Benefits and Pitfalls of using Scrum Software Development Methodology. Cited 27.4.2018, https://www.belatrixsf.com/blog/benefits-pitfalls-of-using-scrum-software-development-methodology/

Ayyagari, M., Beck, T. & Demirgüç-Kunt, A. 2007. Small and medium enterprises across the globe. Small Business Economics: An international journal 29 (4), 415-434.

Barba-Sánchez, V., del Pilar Martínez-Ruiz, M. & Jiménez-Zarco, A. I. 2007. Drivers, Benefits and Challenges of ICT Adoption by Small and Medium Sized Enterprises (SMEs): A Literature Review. Problems and Perspectives in Management 5 (1), 103-114.

Beck, T., Demigüç-Kunt, A. & Levine, R. 2005. SMEs, Growth, and Poverty, Washington D.C.: World Bank Group.

Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. Computer 21(5), 61-72.

Boehm, B. 2000. Spiral Development: Experience, Principles, and Refinements. Los Angeles: University of Southern California.

Brookfield, D., Fischbacher-Smith, D., Mohd-Rahim, F. & Boussabaine, H. 2014. Conceptualising and responding to risk in IT projects. Risk Management 16 (3), 195-230.

BusinessDictionary.com 2018. What is a business? definition and meaning. Cited 16.3.2018, http://www.businessdictionary.com/definition/business.html

BusinessDictionary.com 2018. What is informal sector? definition and meaning. Cited 17.3.2018, http://www.businessdictionary.com/definition/informal-sector.html

Cambridge English Dictionary 2018. software development meaning in the Cambridge English Dictionary. Cited 27.3.2018, https://dictionary.cambridge.org/dictionary/english/software-development

Cambridge English Dictionary 2018. test Meaning in the Cambridge English Dictionary. Cited 30.3.2018, https://dictionary.cambridge.org/dictionary/english/test

Chau, P. 1994. Selection of packaged software in small businesses. European Journal of Information Systems 3 (4), 292-302.

Collins English Dictionary 2018. Business definition and meaning. Cited 8.3.2018, https://www.collinsdictionary.com/dictionary/english/business

European Commission 2003. EUR-Lex - 32003H0361 - EN - EUR-Lex. Cited 16.3.2018, http://eur-lex.europa.eu/eli/reco/2003/361/oj

European Commission 2016. User guide to the SME Definition. Luxembourg: Publications Office of the European Union.

Fayad, M., Laitinen, M. & Ward, R. 2000. Software Engineering in the Small. Communications of the ACM 43 (3), 115-118.

Fink, D. 1998. Guidelines for the Successful Adoption of Information Technology in Small and Medium Enterprises. International Journal of Information Management 18(4), 243-253.

Gordiyenko, S. 2014. Waterfall Software Development Life Cycle (SDLC) Model: Steps, Stages, Case Studies. Cited 7.4.2018, https://xbsoftware.com/blog/software-development-life-cycle-waterfall-model/

Layton, M. 2018. 10 Key Benefits of Scrum. Cited 8.4.2018, http://www.dummies.com/careers/project-management/10-key-benefits-of-scrum/

Maciaszek, L. 2007. Requirements Analysis and System Design. Third Edition. Harlow: Pearson Educated Limited.

Marakas, G. & O'Brien, J. 2011. Management Information Systems. 10th Edition. New York: McGraw-Hill/Irwin.

OECD 2017. Small, Medium, Strong. Trends in SME Performance and Business Conditions. Paris: OECD Publishing.

OSAKO 2018. About OSAKO. Cited 25.2.2018, http://www.osakoweb.fi/en/about-us/

OSAKO 2017a. Student Union OSAKO. Cited 25.2.2018, https://www.oamk.fi/opinto-opas/en/student-activity/student-union-osako

OSAKO 2017b. Field-Specific Student Organisations. Cited 25.2.2018, https://www.oamk.fi/opinto-opas/en/student-activity/field-specific-student-organizations

Pressman, R. & Maxim, B. 2015. Software Engineering: A Practicioner's Approach. 8th Edition. New York: McGraw-Hill.

Project Management Institute 2004. A Guide to the Project Management Body of Knowledge. Upper Darby: PMI Publications.

QuickScrum, 2016. Is Scrum a methodology? Cited 7.4.2018, https://www.quickscrum.com/Article/ArticleDetails/2026/1/Is-Scrum-a-methodology

Radack, S. 2003. Security Consideration in the Information System Development Life Cycle. Gaithersburg: Information Technology Laboratory.

Raz, T. & Michael, E. 2001. Use and benefits of tools for project risk management. International Journal of Project Management 19, 9-17.

Robu, M., 2013. The Dynamic and Importance of SMES in Economy. The USV Annals of Economics and Public Administration 13 (1), 84-89.

Rouse, M. 2016. What is software development? - Definition from WhatIs.com. Cited 1.5.2018, https://whatis.techtarget.com/definition/software-development

Ruparelia, N. 2010. Software Development Lifecycle Models. ACM SIGSOFT Software Engineering Notes 35 (3), 8-13.

Schnidt, R., Lyytinen, K., Keil, M. & Cule, P. 2001. Identifying Software Project Risks: An International Delphi Study. Journal of Management Information Systems 17 (4), 5-36.
Scrum Alliance 2017. State of Scrum 2017-2018. Scrum Alliance.

SCRUMStudy 2017. An Introduction to SCRUM Framework. Cited 8.4.2018, http://blog.scrumstudy.com/an-introduction-to-scrum-framework/

Singh, R., Garg, S. & Deshmukh, S. 2008. Challenges and strategies for competitiveness of SMEs: a case study in the Indian context. Int. J. Services and Operations Management 4 (2), 181-200.

Siroky, D. 2016. The Gold Master: Why Modern Software is Never Finished. Cited 26.3.2018, http://www.plutora.com/blog/gold-master-why-modern-software-is-never-finished

Software Testing Fundamentals 2018a. Software Quality - Software Testing Fundamentals. Cited 30.3.2018, http://softwaretestingfundamentals.com/software-quality

Software Testing Fundamentals 2018b. Software Quality Assurance - Software Testing Fundamentals. Cited 30.3.2018, http://softwaretestingfundamentals.com/software-quality-assurance/

Software Testing Fundamentals 2018c. Software Quality Control - Software Testing Fundamentals. Cited 30.3.2018, http://softwaretestingfundamentals.com/software-quality-control/

Statistics Finland 2018. Small and medium sized enterprises | Concepts. Cited 8.3.2018, https://www.stat.fi/meta/kas/pienet_ja_keski_en.html

Techopedia.com 2018. What is software engineering?. Cited 26.3.2018, https://www.techopedia.com/definition/13296/software-engineering

Techopedia.com 2018. What is the System Development Lifecycle (SDLC)? - Definition from Techopedia. Cited 2.5.2018, https://www.techopedia.com/definition/24776/system-development-lifecycle-sdlc

Techopedia 2018. What is software?. Cited 26.3.2018, https://www.techopedia.com/definition/4356/software

Techopedia.com 2018. What is the Software Development Life Cycle?. Cited 29.3.2018, https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc

Tesch, D., Kloppenborg, T. & Frolick, M. 2007. IT Project Risk Factors: The Project Management Professionals Perspective. The Journal of Computer Information Systems 47 (4), 61-69.

Uhlig, D. 2018. Advantages and Disadvantages of the Scrum Project Management Methodology. Cited 8.4.2018, http://smallbusiness.chron.com/advantages-disadvantages-scrum-project-management-methodology-36099.html

Yrittäjät.fi 2018. The small and medium-sized enterprises. Cited 17.3.2018, https://www.yrittajat.fi/en/about-federation-finnish-enterprises/small-and-medium-sized-enterprises-526261